.NET Framework 4 - ASP.NET

**Walkthrough: Creating an Asynchronous HTTP Handler**

This walkthrough illustrates how to create an asynchronous HTTP handler. Asynchronous HTTP handlers enable you to start an external process (such as a method call to a remote server) while the handler continues processing. The handler can continue without waiting for the external process to finish.

During processing of an asynchronous HTTP handler, ASP.NET puts the thread that would ordinarily be used for the external process back into the thread pool until the handler receives a callback from the external process. This can prevent thread blocking and improve performance, because only a limited number of threads can be executing at the same time. If many users request synchronous HTTP handlers that rely on external processes, the operating system can quickly run out of threads because many threads are blocked and waiting for an external process.

The example in this walkthrough demonstrates an asynchronous HTTP handler that processes requests for files that have the file name extension .SampleAsync in an ASP.NET application. The example shows the code for the handler, and then how to map the .SampleAsync extension to the handler in ASP.NET. The example also shows how to map the .SampleAsync extension to ASP.NET in IIS, so that IIS forwards requests that end in .SampleAsync to ASP.NET.

For more information about how the ASP.NET runtime interacts with IIS 6.0, see [ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/ms178473.aspx). For more information about ASP.NET integration with IIS 7.0, see [ASP.NET Application Life Cycle Overview for IIS 7.0](http://msdn.microsoft.com/en-us/library/bb470252.aspx).

Tasks illustrated in this walkthrough include the following:

* How to create the code for an HTTP handler class. The class must implement the [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method and the [IsReusable](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.isreusable.aspx) property.
* How to register the handler in the Web.config file and map the .SampleAsync file name extension to it.
* How to map the .sample file name extension to ASP.NET in IIS.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifPrerequisites

In order to complete this walkthrough, you will need:

* Visual Studio or Visual Web Developer.
* An ASP.NET Web site that you can run by using IIS.
* IIS 6.0 or IIS 7.0.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCreating an Asynchronous HTTP Handler Class

To begin, you will create a class that implements the asynchronous handler.

**To create the HelloWorldAsyncHandler HTTP handler class**

1. If the Web site that you are working with does not already have an App\_Code folder, create one under the root of the site.
2. In the App\_Code directory, create a class named HelloWorldAsyncHandler and add the following code to the class file:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl34_ctl00_ctl00_code');" \o "Copy Code)

Imports Microsoft.VisualBasic

Imports System.Web

Imports System.Threading

Public Class HelloWorldAsyncHandler

Implements IHttpAsyncHandler

Public ReadOnly Property IsReusable() As Boolean Implements System.Web.IHttpHandler.IsReusable

Get

Return False

End Get

End Property

Public Function BeginProcessRequest( \_

ByVal context As System.Web.HttpContext, \_

ByVal cb As System.AsyncCallback, \_

ByVal extraData As Object) \_

As System.IAsyncResult \_

Implements System.Web.IHttpAsyncHandler.BeginProcessRequest

context.Response.Write("<p>Begin IsThreadPoolThread is " \_

& Thread.CurrentThread.IsThreadPoolThread & "</p>" & vbCrLf)

Dim asynch As New AsynchOperation(cb, context, extraData)

asynch.StartAsyncWork()

Return asynch

End Function

Public Sub EndProcessRequest(ByVal result As \_

System.IAsyncResult) \_

Implements System.Web.IHttpAsyncHandler.EndProcessRequest

End Sub

Public Sub ProcessRequest(ByVal context \_

As System.Web.HttpContext) \_

Implements System.Web.IHttpHandler.ProcessRequest

Throw New InvalidOperationException()

End Sub

End Class

Class AsynchOperation

Implements IAsyncResult

Private \_completed As Boolean

Private \_state As [Object]

Private \_callback As AsyncCallback

Private \_context As HttpContext

ReadOnly Property IsCompleted() As Boolean \_

Implements IAsyncResult.IsCompleted

Get

Return \_completed

End Get

End Property

ReadOnly Property AsyncWaitHandle() As WaitHandle \_

Implements IAsyncResult.AsyncWaitHandle

Get

Return Nothing

End Get

End Property

ReadOnly Property AsyncState() As [Object] \_

Implements IAsyncResult.AsyncState

Get

Return \_state

End Get

End Property

ReadOnly Property CompletedSynchronously() As Boolean \_

Implements IAsyncResult.CompletedSynchronously

Get

Return False

End Get

End Property

Public Sub New(ByVal callback As AsyncCallback, \_

ByVal context As HttpContext, \_

ByVal state As [Object])

\_callback = callback

\_context = context

\_state = state

\_completed = False

End Sub

Public Sub StartAsyncWork()

ThreadPool.QueueUserWorkItem(New WaitCallback(AddressOf StartAsyncTask), Nothing)

End Sub

Private Sub StartAsyncTask(ByVal workItemState As [Object])

\_context.Response.Write("<p>Completion IsThreadPoolThread is " & Thread.CurrentThread.IsThreadPoolThread & "</p>" & vbCrLf)

\_context.Response.Write("Hello World from Async Handler!")

\_completed = True

\_callback(Me)

End Sub 'StartAsyncTask

End Class 'AsynchOperation

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl34_ctl00_ctl01_code');" \o "Copy Code)

using System;

using System.Web;

using System.Threading;

class HelloWorldAsyncHandler : IHttpAsyncHandler

{

public bool IsReusable { get { return false; } }

public HelloWorldAsyncHandler()

{

}

public IAsyncResult BeginProcessRequest(HttpContext context, AsyncCallback cb, Object extraData)

{

context.Response.Write("<p>Begin IsThreadPoolThread is " + Thread.CurrentThread.IsThreadPoolThread + "</p>\r\n");

AsynchOperation asynch = new AsynchOperation(cb, context, extraData);

asynch.StartAsyncWork();

return asynch;

}

public void EndProcessRequest(IAsyncResult result)

{

}

public void ProcessRequest(HttpContext context)

{

throw new InvalidOperationException();

}

}

class AsynchOperation : IAsyncResult

{

private bool \_completed;

private Object \_state;

private AsyncCallback \_callback;

private HttpContext \_context;

bool IAsyncResult.IsCompleted { get { return \_completed; } }

WaitHandle IAsyncResult.AsyncWaitHandle { get { return null; } }

Object IAsyncResult.AsyncState { get { return \_state; } }

bool IAsyncResult.CompletedSynchronously { get { return false; } }

public AsynchOperation(AsyncCallback callback, HttpContext context, Object state)

{

\_callback = callback;

\_context = context;

\_state = state;

\_completed = false;

}

public void StartAsyncWork()

{

ThreadPool.QueueUserWorkItem(new WaitCallback(StartAsyncTask), null);

}

private void StartAsyncTask(Object workItemState)

{

\_context.Response.Write("<p>Completion IsThreadPoolThread is " + Thread.CurrentThread.IsThreadPoolThread + "</p>\r\n");

\_context.Response.Write("Hello World from Async Handler!");

\_completed = true;

\_callback(this);

}

}

The code implements the [BeginProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.beginprocessrequest.aspx) method. The method writes a string to the [Response](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.response.aspx) property of the current [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx) object, creates a new instance of the AsyncOperation class, and calls the StartAsyncWork method. The StartAsyncWork method then adds the StartAsyncTask delegate to the [ThreadPool](http://msdn.microsoft.com/en-us/library/system.threading.threadpool.aspx) object. When a thread becomes available, the StartAsyncTask method is called, which writes out another string to the [Response](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.response.aspx) property. The task then finishes by invoking the [AsyncCallback](http://msdn.microsoft.com/en-us/library/system.asynccallback.aspx) delegate.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifRegistering the Custom HTTP Handler in IIS 6.0

After you have created the custom HTTP handler class, you must register it in the application's Web.config file. This enables ASP.NET to find the handler when requests are made for resources whose URL ends with .SampleAsync.

There are different procedures for registering the handler, depending on whether you are working with IIS 6.0 or IIS 7.0. This section describes how to register a handler in IIS 6.0. The next section describes how to register a handler in IIS 7.0.

**To register the handler in IIS 6.0**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted markup to the Web.config file:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl35_ctl00_ctl00_code');" \o "Copy Code)

<configuration>

<system.web>

<httpHandlers>

<add verb="\*" path="\*.SampleAsync"

type="HelloWorldAsyncHandler"/>

</httpHandlers>

</system.web>

</configuration>

The configuration element registers the HelloWorldAsyncHandler handler as the handler for requests that end with .SampleAsync.

1. Register an application extension mapping for the .SampleAsync file name extension by using IIS Manager. For more information, see [How to: Configure an HTTP Handler Extension in IIS](http://msdn.microsoft.com/en-us/library/bb515343.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifRegistering the Custom HTTP Handler in IIS 7.0

In IIS 7.0, an application can run in either Classic or Integrated mode. In Classic mode, requests are processed much the same way as they are in IIS 6.0. In Integrated mode, IIS 7.0 manages requests by using a pipeline that enables it to share requests, modules, and other features with ASP.NET.

For IIS 7.0, the handler registration requires registering the handler either in the Web.config file or in IIS Manager. Because administration is centralized in IIS 7.0, changes in an application's Web.config file are reflected in the IIS Manager interface for the application and vice versa. In the procedures that follow, the handlers are registered in the Web.config file.

There are different procedures for registering the handler for IIS 7.0 running in Classic mode and running in Integrated mode. Follow the procedure for the IIS mode that you are using.

**To register the handler in IIS 7.0 running in Classic mode**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

|  |
| --- |
| **Note** |
| Substitute the correct path for the aspnet\_isapi.dll file. The .dll file is in the folder where the .NET Framework is installed. By default this is C:\WINDOWS\Microsoft.NET\Framework\*version*. |

1. [Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl36_ctl00_ctl01_code');" \o "Copy Code)
2. <configuration>
3. <system.web>
4. <httpHandlers>
5. <add verb="\*" path="\*.SampleAsync"
6. type="HelloWorldAsyncHandler"/>
7. </httpHandlers>
8. </system.web>
9. <system.webServer>
10. <handlers>
11. <add verb="\*" path="\*.SampleAsync"
12. name="HelloWorldAsyncHandler"
13. type="HelloWorldAsyncHandler"
14. modules="IsapiModule"/>
15. scriptProcessor="%path%\aspnet\_isapi.dll"
16. </handlers>
17. </system.webServer>
18. </configuration>
19. The configuration element registers the custom handler by class name and maps the .SampleAsync file name extension to that handler.

|  |
| --- |
| **Description: NoteNote** |
| Because you are registering a custom file name extension, you register the handler in both the **handlers** section and the **httpHandlers** section. In Classic mode, for backward compatibility, the handler is specified as an ISAPI module by using the **modules** attribute. The path of the ASP.NET ISAPI dll is specified by using the **scriptProcessor** attribute. The **name** attribute is required in the **handlers** section. |

**To register the handler in IIS 7.0 running in Integrated mode**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl36_ctl00_ctl03_code');" \o "Copy Code)

<configuration>

<system.webServer>

<handlers>

<add verb="\*" path="\*.SampleAsync"

name="HelloWorldAsyncHandler"

type="HelloWorldAsyncHandler"/>

</handlers>

</system.webServer>

</configuration>

The configuration element registers the custom handler by class name and maps the .SampleAsync file name extension to that handler.

|  |
| --- |
| **Description: NoteNote** |
| The registration is in the **handlers** section, but not in the **httpHandlers** section. The **name** attribute is required. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifTesting the Custom HTTP Handler

After you have created and registered the custom HTTP handler, you can test it.

**To test your custom HTTP handler**

* Browse to the application and enter a URL in the browser that ends in .SampleAsync.

The text defined in the HelloWorldAsyncHandler class is displayed.